PROGRAMMER'S REFERENCE MANUAL

Print 1

AUTOMATIC CODING SYSTEM

FOR THE IBM 705

This manual is a reprint of the subject matter originally included in the Intermediate Manual, PRINT I with the following changes:

Page No.	*Line No.*	
20	15	E09: Echo check or 0903 error, channel 12 in carriage control tape
23	(added)	op code: ENT; comments: Enter PRINT mode
	(added)	op code: ATR; comments: explanation on p. 34
	last	op code: RCD; variable field: UNIT, TRADD; comments: Read a card. UNIT is either tape t or card reader. Transfer to TRADD on end-of-file condition. (Optional specification of TRADD.)
31	(added)	op code: ENT (inserted between lines 2 and 3 of the example, i.e., between SCALE and RSTRT)
33	24	In the case of arithmetic commands, . . . (to replace: For the MAD, MMA, PMA and MPA instructions, . . .)
37	6	LAR l 2044 2059 2059
52	26	Change △ (word length) + 19/21/23, which refers to first address of TSC, to △ (word length) + 19/23/27
52a through 52y		Omitted

Contents

This manual is a programmer's reference to PRINT I. Further information will be supplied in supplementary form early in 1957. This includes the 10- and 12-digit mantissa systems, the tinkertoy appendix, and a primer to be read in conjunction with this manual.

Existing pre-edit and executive routines will be furnished in card form upon written request, automatically placing those installations on the mailing list for subsequent revisions, particularly to include all mantissa lengths. The symbolic listing of the pre-edit routine will not normally be furnished, except upon special request. PRINT I coding forms and instruction cards may be obtained at Stationery Stores under form numbers 19-6905-0 and 887834, respectively. Assistance in programming and operating the PRINT I system may be obtained from Applied Science representatives.

Working committee

Bemer, R. W.
Fromm, D.
Glans, T. B.
Hira, G. R.
Hoggatt, A.
Krasnow, E.
Levitan, R.
Michels, L.
Williams, F. A.

Programming Research Department
International Business Machines Corporation
590 Madison Avenue
New York 22, N. Y.

PRINT I

Purpose

The PRINT I (PRe-edited INTerpretive) system has been primarily designed to meet the engineering and scientific computing needs of those 705 installations where such work is a secondary computing requirement.

General characteristics

PRINT I is an automatic coding system of the interpretive type, designed to make the 705 itself do the major portion of the coding and clerical work. It is designed for ease of learning and operation by personnel with little or no previous programming experience. It has the following desirable features:

1. *Floating point arithmetic.* The programmer need not concern himself with the position of decimal points throughout calculation. Entry of fixed point numbers and production of fixed point printed output may be made without the operator concerning himself with the fact that internal calculation was in the floating point mode.

2. *Matching mathematical functions.* All functions operate near optimum speed and are computed to an accuracy which is consistent with the arithmetic used. Facility is made for the user to insert his own sub-routines, by using the "tinkertoy" appendix. The library of functions is greatly extended by the floating sub-routine feature, which allows non-standard functions in tape storage to be used as though they were standard functions in core memory.

3. *Variable address and instruction format.* The instructions in this system are of varying length and contain a variable number of specified addresses, depending on the amount of information each instruction must carry. This is consistent with the variable length features which enhance the 705. Coding is done in a variable field, with the multiple addresses and other information separated by commas.

4. *Advanced instruction set.* Many useful combinatorial instructions are incorporated to give greater flexibility to calculations. Among these are vector multiply-adds, polynomial multiply-adds, special operations for convergence testing, indirect address features, counting switches, counting printing instructions and a completely automatic table search operation with an adjustable block feature. All of these are performed by the use of a single instruction.

5. *Index registers.* An incremental type of indexing is used for address modification, substantially reducing the number of program steps to be written, by factors of from 2-1 up to 50-1. Each address may be indexed by the sum of the contents of up to three registers, greatly facilitating internal loops. Index registers may also be used as counters for control purposes, without actually being used for address modification.

6. *Repeat instruction.* This instruction controls automatic repetition of a following instruction, allowing grouped data to be handled with very few instructions. This is advantageous in converting input and output data from fixed decimal to floating and vice versa, in table searching, in matrix calculations, etc. It also permits a secondary form of indexing.

7. *Facility.* PRINT 1 may be thought of as a means of using the 705 as a giant but convenient desk calculator. Elapsed time between problem statement and production of answers can now be a matter of hours, rather than days or weeks. The instruction set is straightforward and restrictions are minor; many logical errors in the written program are automatically detected and typed out to the operator in the form of an error message.

8. *Interpretive system.* PRINT 1 is operated by an executive routine which is always in memory during the running of a problem. This routine fabricates the requisite 705 instructions as it computes, finding the various components in the pattern of the converted PRINT instruction. There is no necessity for developing expert machine language programmers; the intricate coding is already built in. The executive routine, under various options, occupies

from 4000 to 6000 characters in memory (equivalent to 800 to 1200 705 instructions), but experience with the system has shown that for mathematical work one PRINT instruction is the equivalent of about 40 705 instructions. The break-even point is therefore at around 30 PRINT instructions, which is a relatively small program. Interpretation is not generally time-consuming in PRINT, because the repeat instruction enables the following instruction to be performed n times in succession with only a single interpretation. For the remaining $n-1$ times, the instruction operates, in general, even faster than the most expert coder or compiler could generate the program. This statement may appear contradictory unless it is understood that, due to the possibility of selecting the most advantageous fixed locations in memory, certain machine characteristics may be utilized to decrease the operating times. These same routines, if compiled in random memory locations, would be incapable of operating correctly.

705 components

The only components required to operate the PRINT system are the magnetic core memory and sufficient tape units (> 3) to handle expected problem size. An on-line printer and on-line card reader are assumed to be available, although they may be dispensed with by certain modifications to the system.

System components

When operating in the PRINT system, the 705 is for all practical purposes changed to a different machine, that is in a non-physical sense. Certain simulated hardware exists in the system, as:

1. *Index registers.* There are three of these registers. They are addressable by certain instructions for setting and augmenting their contents. They are effectively addressable in the body of other instructions to enable their contents to be used to modify addresses.

2. *Limit registers.* There are three of these, one for each index register. They are for maintaining limits to the contents of the index registers, which are

used for automatic termination of loops of indexed instructions.

3. *Line image.* This is an image in memory of the printer type wheels, such that each of the type wheels is effectively addressable. All printing and error correcting routines associated with printing are automatically associated with this line image.

4. *Heading image.* This is also an image in memory of the printer type wheels, but is used exclusively for heading printed pages of reports in any format the programmer desires. The programmer merely uses two cards in his program to specify this heading format.

5. *Card image.* This is an image in memory of the card columns. All columns are effectively addressable. All card reading, whether from the card reader or tape, enters this area; all card writing, whether on tape or to the card punch, is done from this area.

6. *Fixed symbolic locations.* There are six fixed locations in memory. Although addressed symbolically, they are automatically interpreted as actual addresses:

For numbers (data word length)	*For addresses*
PAC1 (Pseudo-ACcumulator 1)	LAR1 (Location of ARG1)
PAC2 (Pseudo-ACcumulator 2)	LAR2 (Location of ARG2)
ARG1 (ARGument 1)	
ARG2 (ARGument 2)	

PAC1 is the basic component for the multi-address instructions, for which it is the understood address. All arithmetic operations send the result to PAC1 as a secondary result storage or temporary working area. The other locations are mainly pertinent to the table search operations.

Overall mode of operation

The use of PRINT to solve a problem falls into four basic steps. They are described very generally here; the actual details of each of these steps are contained in the full description of each which follows later in the manual.

1. The programmer writes, on the symbolic coding form for this system, a

sequence of PRINT and/or 705 instructions designed to bring in data, do arithmetic and logical operations, and finally prepare and produce output data. He does this knowing the function of each of the PRINT instructions, as described in detail under individual sections.

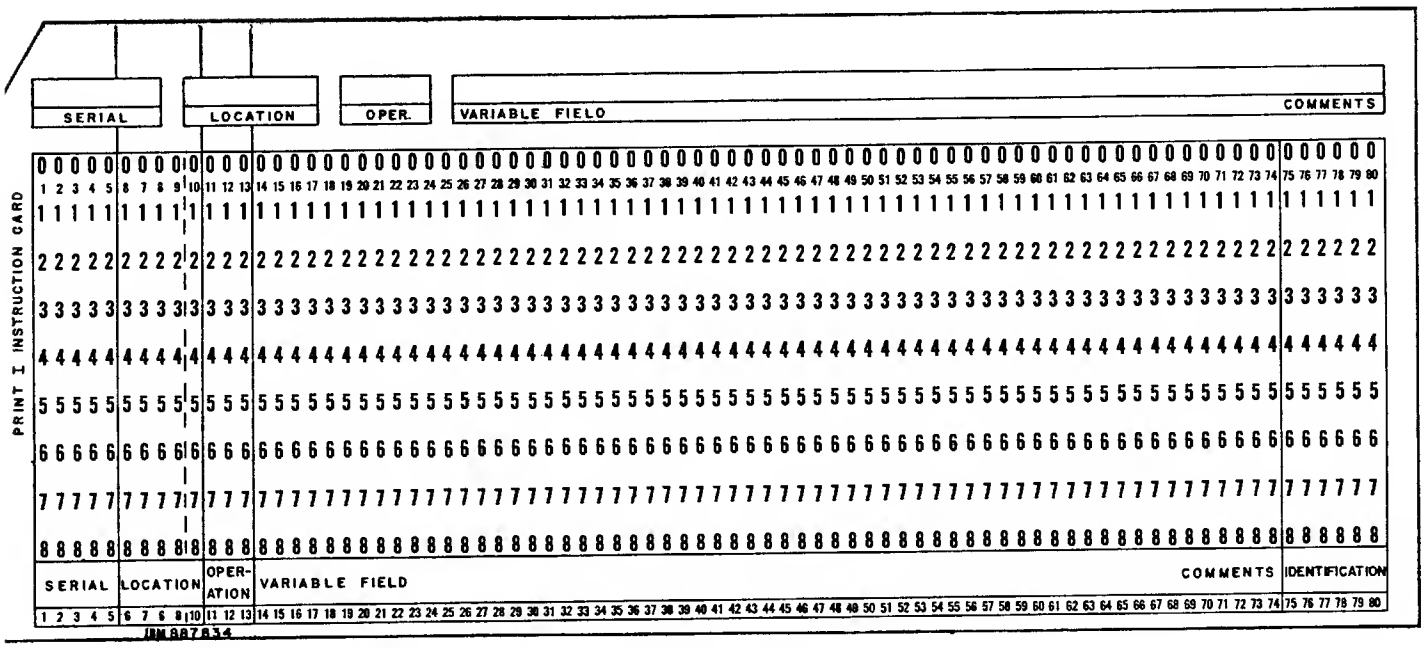
2. Cards are punched from this coding form, each line of coding representing a single card. Punching is done in consecutive columns and may be done without a drum card, as the format is variable. The only column skipped before the end of punching is that defining the end of the variable field and the beginning of the comments.

3. These cards are read into the 705 along with the PRINT 1 system, which consists of two independent parts. The first of these is the pre-edit routine, which will process the program cards and convert them to pseudo-instructions in card or tape form for actual running of the problem. The second part is the executive routine, which is always in core memory during the operation of a program prepared for this system. The pre-edit routine is not maintained in memory after performing its function and is destroyed by entry of the executive routine and the program. It is possible to pre-edit at one time and save the execution of the problem until a later time, as these are entirely separate functions. Pre-editing is a triple function of assembling, compiling and conversion to a form more convenient to the executive routine. For each card with its mnemonic instruction and variable field, pre-edit produces a corresponding pseudo-instruction especially tailored for the fabrication of 705 instructions from its components. These are of varying length of characters according to the operation specified. Matched sets of mnemonic and pseudo-instructions may be printed at pre-edit time, at the option of the operator, together with the comments punched in the right hand part of the variable field. This should be his permanent coding record.

4. The actual running of the problem is under the control of the executive routine, which may be called from tape immediately after pre-editing. The executive routine fills from 4000 to 6000 characters in memory, including the floating sub-routine position and input-output images. Overflow or sign check indicators are not used in PRINT as decision elements. They are reserved for stops while operating with 705 instructions and the switches may therefore be set to automatic stop during the operation of PRINT. Any entry to PRINT sets up the ASU's as required for its operation. All ASU's are therefore available for use in 705 language. Their settings should be noted from the ENTer sub-routine (see Page 12) to avoid redundant resetting for 705 usage.

General coding instructions

Addressing in the PRINT 1 system is entirely symbolic; that is, the address nomenclature can be descriptive of the contents. The symbolic address of a location must be a sequence of one alphabetic character followed by three or four alphanumeric characters. If these following characters are all numeric the address is said to be "regional", which is a sub-class of symbolic notation with certain useful properties (see the next section). A "region" is indentifiable by the first or first two characters (i.e., the "G" region, the "F3" region). If the sequence begins with a numeric character, the following characters must be all numeric and this signifies an actual 705 address. The symbolic locations may be coded in any sequence desired. The format of the PRINT coding card is:



Columns 1-5 — *Serial number.* This provides for sequence control and the collation of change cards. Serial numbers must be in ascending sequence. A convenient convention is the use of the first two columns for coding page number, the second two for line number and the last for inserts.

Columns 6-10 — *Symbolic location.* This field provides a referral name for the entry; that is, if the entry is not referred to by any other instruction in the program, the

field can and should be left unpunched. This will reduce the size of the table of symbolic and actual address correspondence, thus decreasing the running time of pre-editing by minimizing search time. If the field is punched it must follow the rules for a symbolic address, with the alphabetic character in column 6. Punching is optional in column 10.

Columns 11-13 — *Operation code.* This field must be punched with a 3 character (including blanks as characters) mnemonic code which describes the function of the entry. This may be a PRINT operation, a 705 operation or one of the several special operations for constants, memory reservations, origins or headings.

Columns 14-74 — *Variable field.* This field is punched as the requirements of the particular instruction dictate. The first blank column indicates the beginning of the comments field, which may actually extend through to column 80 if no identification is required.

Columns 75-80 — *Identification.* Any 6 character alphanumeric designation may be punched (ganged) here to identify the program. An identification obtained from the *first* card of the symbolic program deck will be punched in the first 6 columns of the 705 load cards produced by pre-edit for reloading, and will also appear in the heading of the pre-edit listing.

Regional notation

An alternate method of using symbolic addressing is available if the programmer desires to code with "unitized" components. If the symbolic address is regional, the serial number in columns 1 to 5 may be omitted, in which case the numbering sequence within the regional address controls instruction sequence in the program. Columns 6 to 10 will always be filled with a regional address, regardless of referral status, and referral will now be indicated by punching an 11-punch in column 1. The drum card of the keypunch should be arranged to skip to column 6 for the next punching. If the programmer fears becoming careless in noting referral addresses, he may gang the 11-punch in column 1 of every card, but this could retard the pre-edit process by carrying a complete table of referrals.

Regional addressing is convenient for quick replacement of identifiable components with a specific function in the program. To illustrate, consider that in a program to compute airplane performance the calculation of engine

thrust is assigned to region T1. This region receives as input data certain information produced by other regions, computes thrust with this data and certain equations, finally putting this resultant thrust value in a location usable to other regions. For several different engines, or several different ways of computing thrust, different T1 regions would be coded. All of these receive and store data in addresses not common to the computing regions, but accessible to all. If the programmer wishes to compute performance for a certain configuration he selects one form for each region involved and processes this combination through the pre-edit routine. He is thus guaranteed that housekeeping is perfect and that no pattern of computation will have been erroneously disrupted.

Coding PRINT instructions

The variable field of PRINT instructions is coded according to the context of the instructions. Each operation is described as having a certain number of positions in the variable field. Each of these positions are separated by commas. An exception occurs for indexable instructions, where a position is defined to contain both the address and its index register tag, although separated by a comma. The tag is therefore in a sub-position immediately following the address it modifies. An address is a field of four or five characters. It is symbolic if it begins with an alphabetic character; if the first character is numeric, all must be numeric and the address is actual. An index tag is a field composed of the digits 1, 2 and 3 not repeated which designate the index register or registers which are to affect the address in context. If an address is not to be indexed, no tag field is written. In the first example the address in the second position is the only one indexed; in the second example the address in the third position is also indexed.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
	ADD	P 202, R 532, 12, QYR5	A blank column terminates
	ADD	P 202, R 532, 12, QYR5, 2	the instruction and starts
	MPY	RATE, 2, TIME, 23, DIST, 3	the comments

Both numeric and alphabetic characters are used in coding for this system.

As a standard precautionary practice, always write the letters Ø, I and Ƶ as shown, with slashes and cross-bars to safely distinguish them from the numerals 0, 1 and 2.

Every program will begin with either a 705 instruction or an ENT (ENTer) instruction. Every transfer of control from 705 to PRINT instructions and back will be called for by the programmer. Consequently, every block of PRINT instructions must be preceded by an ENT, which is compiled by the pre-edit into three 705 instructions:

LOCATION 6. .10	OPERATION CODE 11. .13	VARIABLE FIELD 14.	COMMENTS .80
BADD-11	SET	4, 1	
BADD-6	LØD	BADD-6, 1	
BADD-1	TR	(to the address of the first instruction in the ENT Sub-routine in	
		the PRINT Executive Routine)	

The basic address of the first PRINT instruction is at BADD. This is computed by the ENT sub-routine from BADD-6 in ASU01. ASUs are set to length and control is transferred to the fetch sub-routine, which brings in the first PRINT instruction in the interpretation cycle.

Following an ENT, all entries are considered by pre-edit as PRINT instructions until the instruction LVE (LeaVE) is encountered. LVE is a PRINT instruction whose address is normally pre-edited as the location of the next 705 instruction. When executed, it will cause the executive routine to transfer control to that instruction. All succeeding entries will then be considered as 705 operations or special operations until another ENT is encountered. Thus ENT and LVE are normally coded without addresses in the variable field. When LVE is coded with the address of a 705 instruction, pre-edit gives that address in conversion rather than that of the next 705 instruction in sequence.

An asterisk in the variable field of an ENT indicates that this is the point at which the operation of the program will be commenced, rather than the first ENT or 705 instruction encountered by the pre-edit. If more than one ENT contains an asterisk in the variable field, the first encountered takes precedence. An ENT must not precede a 705 or special instruction, else a compiling error will occur in memory assignment. When successive entries change from PRINT to 705 instructions or vice versa, without intervening ENT or LVE entries, pre-edit will type out a mode change error message.

Upon executing a LVE instruction, advantage may be taken in 705 operations of the fact that the ASUs are left with known length settings, as follows:

ASU	*Length*	*ASU*	*Length*	*ASU*	*Length*
01	4	06	4	11	2
02	1	07	4	12	3
03	2	08	Word Length	13	Indeterminate
04	4	09	1	14	5
05	4	10	1	15	18

Two successive commas imply that the intervening address is that of the main pseudo-accumulator PAC1, which is a field in memory reserved for this function. PAC1 is incapable of being indexed, even if tagged; a zero indicator is automatically inserted for it by the pre-edit routine. PAC1 may also be addressed by the symbol PAC1. If fewer addresses are coded than required by a particular instruction, the remaining addresses will be interpreted to be PAC1 by the pre-edit. For example, the following instructions are equivalent, incidentally doubling the contents of PAC1.

ADD PAC1, PAC1, PAC1 ADD , , b ADD

An exception to this rule occurs in the SAC operation. If the result addresses are not specified in the second and third positions of the variable field, the second position is interpreted as PAC1 and the third as PAC2.

Coding 705 instructions

705 instructions are coded either before the first ENT or between LVE and the next ENT. Standard 705 mnemonic codes are used. The first field after the operation code is interpreted as the address. An actual address can be any combination of 4 or 5 numeric digits, as the leading zero does not have

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
	TR	CYCLE	
	RAD	FØL3-52, 2	FØL3-52 zoned for ASU 02
	SET	9, 13	Set ASU 13 to length of 0009

to be punched. A symbolic address must satisfy the same criteria as the addresses of PRINT instructions do. If the address is terminated in a sign, the next field is interpreted as an increment. The following field is the ASU designation. The instruction refers to the 00 accumulator if no ASU coding is present.

Special operations

PRINT 1 uses various mnemonic special operation codes for initial organization of a program. These are illustrated at the end of this section. These operations are static and do not create working PRINT or 705 instructions.

ADC (ADdress Constant) produces a 4 character constant which is the 705 address determined by the symbolic address, increment and ASU coding in the variable field.

ORG (ORiGin) controls the actual memory assignment of subsequent instructions or areas. These are four types of ORG entries, and pre-edit will handle up to 100 ORGs with addresses in the variable field.

1. When the address in the variable field is actual (i.e. numeric), the first character of the next entry will be at that specified address.
2. When the variable field is blank, all following entries will be assigned in order following the highest location assigned previously.
3. When the address in the variable field is identical to the symbolic address of the ORG itself, the location of the previous entry will be stored in the table of origins for later reference.
4. When the address in the variable field is a symbolic address stored under the conditions of type 3, the succeeding entries will be assigned following the location stored by type 3. This is a device for remembering and continuing an interrupted series.

CON (CONstant) and BLK (BLocK) reserve filled or unfilled memory space. For either entry, the first position in the variable field is a number of from 1 to 3 digits specifying the length of the entry, which in the case of CON is limited to 50 characters. If an asterisk precedes the length specification the entry starts with a memory position ending in 0 or 5. If a constant is signed, the plus or minus sign follows just after the length; plus

signs may not be omitted. A blank column and the actual constant follow. A record or group mark may appear only in the first character of a constant. The address assigned by pre-edit is that of the highest memory position (or right-hand character) in the field.

FLC (FLoating Constant) is a PRINT entry corresponding to CON for the 705. Coded in the variable field is the sign and the 1 or 2 character exponent, followed by the mantissa sign and as many digits of the mantissa as the coder cares to write. These are not separated by commas. The initial number of characters in the mantissa is not limited; pre-edit automatically converts to internal format, without rounding if the number of characters exceeds mantissa length for the system. A blank variable field is considered an error by pre-edit.

DEL (DELete) is a special operation for program correction and is explained in the operation of pre-edit and system entry.

REG (REGister reservation) is a PRINT entry corresponding to BLK for 705 entries. It is used to reserve memory space for floating point PRINT numbers (words). For reservation of a single word or number space the variable field is normally left blank, as the length is already specified by the system. The word length area is addressed in other instructions by the symbolic address of the REG entry. If the address is regional, a lower address in that same region may be written in the variable field, signifying reservation for all addresses within those limits. For indexing purposes, all randomly symbolic addresses in an operational sequence must be reserved sequentially and individually. It is therefore preferable to reserve addresses for indexable instructions in the regional mode. Pre-edit will accommodate up to 60 of these multiple reservations.

SAY (SAY it) will enter a line of comment into the pre-edit listing.

HDG (HeadDinG) is a PRINT entry for inserting a page heading for printed reports. Coded in the variable field are a blank (column 14) and up to 60 characters in columns 15 to 74. One or two cards may be used, the second corresponding to type wheels 61 to 120.

FIN (FINish) is an entry which signifies termination, in the card reader, of the program to be pre-edited. It has the same effect as an end-of-file signal. This permits data cards to be loaded into the reader simultaneously with the program, without being considered as entries to be pre-edited. The entire program may thus be run in a continuous fashion.

PRINT I SYMBOLIC CODING FORM

PROBLEM: ILLUSTRATING THE VARIOUS USAGES FOR SPECIAL OPERATIONS

CODER | DATE | PAGE OF

SERIAL (1-5)	LOCATION (6-10)	OPERATION CODE (11-13)	VARIABLE FIELD (14-)	COMMENTS (-80)
	MANT	ADC	PØWER - 2,9	(,09 also correct)
		ØRG	5040	(05040 also correct)
	A123	RAD	CØL08,2	(instruction located at 5044)
		ØRG	35040	
	CØN45	CØN	2+b45b	(the number +45, located at address 35041)
	XNYC	BLK	56	(reserves 56 characters in memory)
		CØN	*28bFOURbSCOREb	
	ZERØ	FLC	+0+0	0×10^{0} = ZERO
	LØC1	FLC	+1+1	$.1 \times 10^{1}$ = ØNE
		FLC	-12+528	$.528 \times 10^{-12}$
		FLC	+3-2	$-.2 \times 10^{3} = -200$
		FLC	+3-200	"
10000		DEL	10006	
	F121	REG	F119	
	F119	REG	F121	
	F119	REG		Identical effects
	F120	REG		
	F121	REG		
	F121	REG		Will cause normal indexing
	F120	REG		to occur in
	F119	REG		reverse order
		SAY	THE FØLLØWING 3 ADDRESSES ARE DESIGNED FØR INDEXING	
	JØNES	REG		Pre-edit assigns memory positions
	SMITH	REG		in the order in which
	BRØWN	REG		they are encountered
	A001A	REG	A100A	Useful technique for greatly
	A001B	REG	A100B	expanding the number
	B001A	REG	B100A	of available regions
		SAY	THE FØLLØWING LINES SHØW ILLEGAL USAGE	
10006		DEL	10001	Will delete 10006 only
20684		DEL	A106,13,,F*/4	(Dangerous to maintain old variable field)
	F120 1	REG		(Will not be included in F119 - F121 sequence)
		REG	SMITH	
	SMITH	REG	JØNES	
	TEMP2	REG	TEMP1	

Indexing

A system of indexing is simulated within the PRINT framework. Most 705 programmers are already familiar with one means of specifying the location of a number without using the actual address. This is symbolic addressing, where the actual address is determined by searching a table of symbolic addresses, each of which has a corresponding actual address. Indexable addressing is one further step up conceptually. If either a symbolic or actual address, not only is the corresponding actual address determined from the symbolic, but the address which the instruction really refers to is that actual address plus the number contained in the index register specified. If that index register has a different number in it every time that the same instruction refers to it, then the same instruction obviously uses a different address every time, although the instruction itself never changes. The examples in Appendix II show how the same instruction may be used repetitively to advantage. The justification for indexing is the resultant economy in the number of instructions that the programmer must write.

In the out-of-context example shown, the angle whose sine is placed in PAC1, and whose cosine is placed in PAC2, is not the angle in the address P220. Since R2 (register 2) and R3 have been set to 3 and 8 word lengths respectively, it is rather the angle in address P231, which is P220 plus 3 plus 8.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -8D
P240	REG	P220	Reserve sequential addresses
	ENT		
	SR 2	3	Set R2 to 3 word lengths
	SR 3	8	Set R3 to 8 word lengths
	SAC	P220, 23	Sine to PAC 1, cosine to PAC 2

The 23 tag after the address in the first position indicated that the address was to be incremented by the sum of the contents of R2 and R3. In PRINT I, the contents are added to the address and indexing is said to be incremental. By contrast, 704 indexing is decremental. There are 3 index registers, referred to as R1, R2 and R3. Any address in an indexable instruction may be incremented by the contents of any of these registers or the arithmetic sum of

the contents of any two or all three. This alteration takes place in a work area before fabricating the necessary machine language instructions from the address portions of the PRINT instruction involved. The original PRINT instruction in operating sequence is never altered. Loops formed by transfer on index instructions will therefore be re-indexed from the original instructions. Such transfer is dependent upon the contents of the index register not having exceeded a specified limit.

The contents of index registers are used only for address modification with 705 add-to-memory instructions. The contents are unsigned and 4 digits in length. Increments are carried in memory as true numbers, decrements as the complements of 40,000. To increase operating speeds, all possible sums of contents of index registers are carried along in memory. When any register is altered, the contents of each combination in which that register participates are altered by the same amount. This permits indexing any address by a single add-to-memory instruction.

Direct access to the contents and limits of index registers may be had (in 705 machine language) by using the actual addresses of 4-character unsigned numeric fields as follows:

R1	0718	R1 limit	0723
R2	0728	R2 limit	0733
R1+R2	0738		
R3	0748	R3 limit	0753
R1+R3	0758		
R2+R3	0768		
R1+R2+R3	0778		

Diagnostic routines

Two types of diagnostic methods for error finding are used with PRINT instructions. The first of these is a memory print associated with the system control, the operation of which is described under the section on pre-editing (page 44). This method is used primarily to determine cause of machine stops during computation.

The second type of diagnostic is that commonly called "snapshot", and is entirely under the selective control of the programmer. This is accomplished by inserting extra instructions in the program to be pre-edited. These instructions are designed by the programmer to view selected intermediate results

or logical path indications. When the program is ascertained to be correct, these snapshot instructions have their operation codes changed to DEL and are re-collated in with the previously assembled program, thus removing them from the operating program. This feature is possible only because of the fast re-assembly time in the PRINT system. Most programs will take from 30 seconds to 1 minute for tape re-assembly.

For detail work, a 705 machine language tracing routine is furnished. This routine (primarily developed by Mrs. Helen Meek of the Hughes Aircraft Company, Culver City, California) may be used as a separate diagnostic tool for all work encountered by the installation, including commercial problems. The basic principle of this routine is the temporary displacement (and storage for later replacement) of certain operating instructions in the working program by transfers to the tracing routine. This permits high speed operation to various points of interest, at which time detailed tracing occurs. High speed operation of the program may be resumed at specified points. Determination of the local regions to be traced is under card control.

Arithmetic operations

All PRINT 1 arithmetic operations use numbers in floating decimal form as the operands. All 705 operations are in fixed decimal form. A floating decimal number is essentially a piece of data and is referred to as a "word". This floating point word is comprised of two parts, a proper decimal fraction (called the "mantissa") with a non-zero leading digit and a power of 10 multiplying that fractional number (called the "power", although "characteristic" is an alternate term). Floating point words are

stored in memory as:	although written for input as FLCs:
$XXX. \ldots \overset{\pm}{X}\overset{\pm}{X}PP$	$\pm PP \pm XXX. \ldots$

The X's represent the digits of the mantissa and the P's represent the two digits of the power, which may range from -99 to $+99$. The dots signify that PRINT 1 is furnished in separate forms for 8, 10 and 12 digit mantissas. The 12 digit system (word length = 14 characters) will be furnished originally with 12 digit arithmetic but having the mathematical functions normally provided with the 10 digit system. A 12 digit system to consistent accuracy and a 20 digit system will be available about January 1957.

Each of these systems will then be complete in itself for all operations,

having all sub-routines designed to an accuracy equivalent to the length of the mantissa. Sample words in input format, floating point format and their equivalent fixed decimal form are:

Input format	*Internal format*	*Actual number*
+0+12345678	+ + 1234567800	.12345678
+5−1234567899	− + 1234567805	12345.678−
−5+1234567899	+ − 1234567805	.0000012345678
+1+6	+ + 6000000001	6
+8+6	+ + 6000000008	60,000,000
−10+6	+ − 6000000010	.00000000006
+6+1	+ + 1000000006	100,000 ($=10^5$)
+2−3579	− + 3579000002	35.79−
+0+0	+ + 0000000000	0

The second example is −.12345678 times 10^5. It can be seen from the examples that when the power is positive, it represents the number of whole number digits; when the power is negative, it represents the number of zeros to be placed after the decimal point before the actual number begins. A power of zero means that the number is a decimal number just as it is without using the power. A true zero is always signed positively.

Whereas as 705 instructions refer to the contents of a single address, PRINT instructions are in a multi-address form. All PRINT operations except FPR are performed without rounding, to save operation time. If this should ever cause inconvenience, use a system with a longer mantissa. Arithmetic instructions which are found to refer to zero operands will operate in accelerated fashion, since all operands are first tested for zero in the arithmetic subroutines.

If an error occurs during execution because of the impossibility of foreseeing certain illegal conditions, an error message will be written on the typewriter. The "tinkertoy" appendix will provide options for this type-out.

If the programmer wishes to conserve memory space he may select the option which types out the letter E followed by a 2 digit code number; referral to the manual will tell him the type of error which has occurred. If economy of memory is not vital, he may select the option of typing out an expository message. Under either option, the actual address of the failing instruction is also typed out. Error messages are:

E01: Division by zero
E02: Logarithm of zero or a negative number
E03: Sine and cosine of an angle greater than $\pm 318\pi$
E04: Square root of a negative number
E05: Power overflow (>99)
E06: Power underflow (<-99) (only if desired by user)
E07: Line image overflow
E08: Too many whole numbers
E09: Echo check, or 0903 error, channel 12 in carriage control tape
E10: Line or HDG won't write correctly 0902
E11: Read card error or card punch error
E12: Card won't punch correctly 0902
E13: 0901 error on write tape
E14: Tape won't read/write correctly
E15: End-of-file before read/write tape completed
E16: Exponential to the base 10 of $|ARG| \geq 99$
Exponential to the base e of $|ARG| \geq 225.65334$

Summary of mnemonic codes

Non-indexable operations

ATR	Alternating TRansfer
BSi	BackSpace tape "i"
LVE	LeaVE PRINT
RCD	Read a CarD
RPL	RePLace
RPT	RePeaT
RWi	ReWind tape "i"
RWR	Repeat With Reset (PACI)
SRi	Set index Register "i"
TMi	write Tape Mark on tape "i"
TNi	Transfer Not testing limit, augmenting "i"
TNZ	Transfer on Non-Zero
TRM	TRansfer on Minus
TRP	TRansfer on Plus
TRU	TRansfer Unconditionally
TRZ	TRansfer on Zero
TXi	Transfer testing indeX limit, augmenting "i"
WCD	Write a CarD
WHi	Write a Heading, space "i"
WLi	Write a Line, space "i"
XTP	eXTract Power

Special operations

ADC	ADdress Constant
BLK	BLocK
CON	CONstant
DEL	DELete
FIN	FINish
FLC	FLoating Constant
HDG	HeaDinG
ORG	ORiGin
REG	REGister reservation
SAY	SAY it

Indexable operations

ADD	ADD
ART	ARcTangent
DIV	DIVide
EXD	EXponential, Decimal base
EXE	EXponential, base E (e)
FLO	FLOat
FPR	Fix for Printing Rounded
FXP	FiX for Printing
LGD	LoGarithm to Decimal base
LGE	LoGarithm to base E (e)
MAD	Multiply – ADd
MDV	Minus DiVide
MMA	Minus Multiply – Add
MMY	Minus MultiplY
MPM	Minus Polynomial Mult.–add
MPY	MultiPlY
PMA	Polynomial Multiply–Add
RTi	Read Tape "i"
SAC	Sine And Cosine
SQR	SQuare Root
SUB	SUBtract
TAB	Transmit ABsolute
TMT	TransMiT
TNA	Transmit Negative Absolute
TRC	TRansfer on Comparison
TRE	TRansfer on Equality
TSC	Table Search on Comparison
WTi	Write Tape "i"

Summary of indexable operations

OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
ADD	ØPER 1, ØPER 2, SUMM	(ØPER 1) + (ØPER 2) ⟶ SUMM
SUB	ØPER 1, ØPER 2, DIFF	(ØPER 1) - (ØPER 2) ⟶ DIFF
MPY	MLPLR, MCAND, PRDCT	(MLPLR) (MCAND) ⟶ PRDCT
MMY	MLPLR, MCAND, NGPRD	-(MLPLR) (MCAND) ⟶ NGPRD
DIV	DVDND, DVSØR, QUØT	(DVDND) ÷ (DVSØR) ⟶ QUØT
MDV	DVDND, DVSØR, NGQUØ	-(DVDND) ÷ (DVSØR) ⟶ NGQUØ
MAD	MLPLR, MCAND, CRSFT	(MLPLR)(MCAND) + (PAC 1) ⟶ CRSFT
MMA	MLPLR, MCAND, CRSFT	-(MLPLR)(MCAND) + (PAC 1) ⟶ CRSFT
PMA	ADDND, MCAND, RSULT	(ADDND) + (PAC 1) (MCAND) ⟶ RSULT
MPM	ADDND, MCAND, RSULT	(ADDND) - (PAC 1) (MCAND) ⟶ RSULT
SQR	SXTY4, EIGHT	$\sqrt{(SXTY4)}$ ⟶ EIGHT
SAC	ANGLE, SINE, CØSIN	sin (ANGLE) ⟶ SINE, cos (ANGLE) ⟶ CØSIN
ART	TNGNT, ANGLE	$\tan^{-1}$ (TNGNT) ⟶ ANGLE
LGD	NUMBR, DECLG	$\log_{10}$ (NUMBR) ⟶ DECLG
LGE	NUMBR, NATLG	$\log_e$ (NUMBR) ⟶ NATLG
EXD	EXPØN, TEN2X	antilog (EXPØN) ⟶ TEN2X
EXE	EXPØN, E2THX	antilog (EXPØN) ⟶ E2THX
(FSR)	ARGUM, RSULT	function (ARGUM) ⟶ RSULT
TMT	HERE, THERE	(HERE) ⟶ THERE
TAB	MINUS, PLUS	|(MINUS)| ⟶ PLUS
TNA	PL/MN, MINUS	|(PL/MN)| ⟶ MINUS
TRC	TRADD, THIS, THAT	Transfer to TRADD if (THIS) ≥ (THAT)
TRE	TRADD, THIS, THAT	Transfer to TRADD if (THIS) = (THAT)
TSC	± Δ, TABLE, ARGUM	Search argument table for first number ≥ (ARGUM), be-
		ginning at TABLE. f(TABLE) is ± Δ word lengths away.
WTi	BEGIN, ENDD, TRADD, TM	Write all successive words from BEGIN to ENDD, inclus-
		ive, as 1 record on tape i. Transfer to TRADD if end-of-
		file is reached, write tape mark if TM is written.
RTi	START, TRADD	Read record from tape i, filling as many successive locat-
		ions as on record, beginning with START. Transfer to
		TRADD if a tape mark is encountered.
FXP	FLNUM, t, wW, dD, s	Fix (FLNUM) x 10^s for print in line image, decimal point
		in type wheel t, with w whole numbers and d decimals.
FPR	FLNUM, t, wW, dD, s	Same as FXP, except round the number when fixing.
FLØ	CØLXX, n, R/L s, FLNUM	Take the n digit number with units position in column XX.
		Move the decimal point R(ight) or L(eft) s positions. Put
		in floating point format in FLNUM.

Summary of non-indexable operations

OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
TRZ	TRADD, TEST	Transfer to TRADD if (TEST) are zero
TNZ	TRADD, TEST	Transfer to TRADD if (TEST) are non-zero
TRP	TRADD, TEST	Transfer to TRADD if (TEST are plus
TRM	TRADD, TEST	Transfer to TRADD if (TEST) are minus
TRU	TRADD	Transfer to TRADD unconditionally
RPL	ADDR[1], INSTR	Replace the 1st address in INSTR by ADDR1
XTP	FIRST, SECND	Give (SECND) the same power as (FIRST)
SRi	± n, ± lim	Set contents of R_i to ±n, limit to ± lim
TNi	TRADD, ± Δ	Augment R_i by ± Δ, transfer to TRADD
TXi	TRADD, ± Δ	Augment R_i by ± Δ, transfer to TRADD only if new (R_i) < lim_i. Otherwise proceed.
RPT	n, ± i, ± j, ± k	Repeat (perform) the next instruction n times, indexing its 1st, 2nd, and 3rd addresses, as they exist, by i, j, and k words lengths respectively.
RWR	n, ± i, ± j, ± k	Reset PAC1 to zero, then operate same as RPT. ± i, ± j and ± k may all be prefaced in RPT and RWR by an * to indicate indexing by number of characters, not word lengths.
ENT		Enter PRINT mode
LVE	TRADD	Leave PRINT. Next instruction is next 705 instruction if TRADD is not written, TRADD if written.
BSi	n	Backspace tape i for n records.
RWi		Rewind tape i.
TMi		Write a tape mark on tape i.
WLi	UNIT, n, TRADD	Write a line, UNIT is tape t or printer. i is the space control after writing. n, TRADD is optional. Write n lines, transfer to TRADD rather than write the (n + 1)th line.
WHi	UNIT, n, TRADD	Write a heading. (Equivalent to WLi).
WCD	UNIT	Write a card. UNIT is either tape t or punch.
RCD	UNIT, TRADD	Read a card. UNIT is either tape t or card reader. Transfer to TRADD on end-of-file condition. (Optional specification of TRADD.)
ATR	FIRST, a, SECND, b	For explanation, see p. 34

Indexable computing operations

Arithmetic operations

These operations are largely self-explanatory from the operation summary preceding this section. It should be noted that MAD, MMA, PMA and MPM are compound, or double, arithmetic operations, although they are still written with only three positions in the variable field. The understood operand is always the contents of PAC1, the primary pseudo-accumulator. Although these accumulative operations may be used singly, their design purpose is for repetitive arithmetic. As such, the result of each operation may be found in PAC1 as well as in the normal result address. The Multiply-ADds are designed for vector products. The Polynomial-Multiply-Adds are designed for evaluation of polynomials with the argument addressed in the second position. Although no index register modification is shown in the summary, all of these operations may have a sub-position for each address, indicating this.

Mathematical function operations

These operations are also self-explanatory. SAC (Sine And Cosine) is the only operation with three positions in the variable field, all others having two positions. Each position may have a sub-position for index register indication. The first position address for all of these operations is that of the argument. In contrast to arithmetic operations, the results are not sent to PAC1 unless specified.

Data transmission operations

TMT (TransMiT), TAB (Transmit ABsolute), and TNA (Transmit Negative Absolute) are operations for moving blocks of data from one group of locations to another. The address in the first position of the variable field is that of the original location; the address in the second position is that of the location to which the data is moved. Both positions may have sub-positions for index register modification. Unless PAC1 is specified in either position

it will be unaffected by the transmittal. Unless destroyed by a later operation, the original contents will be unaffected. TAB guarantees that the contents will be positively signed in the new location, TNA that they will be negatively signed. The first example shows the 40 numbers in locations M001 through M040 being transmitted in a reverse fashion, with a blank location between each number, to the locations P080 down to P002. The second example shows that ANY word-length block of characters may be transmitted by use of this instruction.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
BLNK S	CØN	(word length)	
P080	REG	P001	
	ENT		
	RPT	40, 1, -2	
	TMT	M001, P080	
	TMT	BLNKS, AREA	

Comparison transfer operations

TRC (TRansfer on Comparison) and TRE (TRansfer on Equality) are conditional transfer instructions which make an algebraic comparison of two operands. They are written with three positions in the variable field, the first of which is the address to be transferred to if the condition is met. TRC takes place when the number addressed in the second position is equal to or algebraically greater than the number addressed in the third position. TRE takes place only when these two numbers are equal. All three addresses may be modified by index registers.

These operations have special characteristics when preceded by a RPT or RWR operation. The number of repetitions may be set at a maximum by a positive number or to an indefinite repeat by a negative number in the first position of the RPT instruction. In either case, transfer may occur before the repeat tally is reduced to zero in normal fashion. The tally is therefore automatically reset to zero on a transfer. Considering for purposes of identification that the general instruction is:

TRC/TRE TRADD, THIS, THAT

transfer to TRADD will occur when the contents of THIS, as indexed by the RPT, are greater than or equal to the contents of THAT, as also indexed. When the transfer occurs, the following quantities are left in specialized symbolic locations:

Location	*Contents*
ARG1 (working position)	Last THIS used
ARG2 " "	Last THAT used
LAR1 (Location of ARG1)	Address of last THIS used
LAR2 (Location of ARG2)	Address of last THAT used

LAR1 and LAR2 are usable only by the RPL operation. Using LAR1 or LAR2 as an address in any other instruction will cause an error message in pre-edit. None of these special addresses is indexable by either index registers or RPT or RWR instruction increments. Their index indicators are automatically set to zero by pre-edit. RPT or RWR increments, if used, must be coded as zero by the programmer or a machine stop will occur. When TRC or TRE is used with RPT or RWR the second position in RPT or RWR, which would normally be considered to modify the transfer address, must be coded as zero by the programmer or a machine stop will occur. Although TRC and TRE are indexable instructions, transfer addresses are obviously not indexable in a system using variable length instructions.

Table search operations

TSC (Table Search on Comparison) is a special variation of TRC which is especially designed for fast and flexible table search. Rather than a transfer address, the first position in the variable field contains the differential number of word lengths between the arguments of the table and the corresponding functions of these arguments. No transfer is made after TSC; the next instruction in sequence is executed.

A special RPT or RWR instruction must precede TSC. The first position contains a negative number for indefinte repetition. The second and fourth positions must contain zeros. The third position contains the interval of gross search. Table Search automatically consists of two parts. Letting N symbolize the gross search interval, the first part compares the first argument and successive arguments in intervals of N against the test argument. When one of these is found to equal or exceed the test argument, the search auto-

matically backs up to the previous grouped argument. The second part of the search consists of a comparison of successive arguments in this localized area against the test argument in intervals of *one*. N may be the integer 1 or any other integer, PROVIDED that the number of arguments in the table equals (some multiple of this integer plus one). For example, consider the case of a table with 65 arguments. Valid values of N would be 16, 8, 4 and 2. The first, 17th, 33rd, etc. arguments would be compared against the test arguments if N were assigned as 16. In practice, the most effective interval of gross search is that which most closely approximates the square root of the number of arguments in the table, in this case, 8.

Further suppose that the 33rd argument was found to exceed the test argument. Comparison is now made to the 18th, 19th, 20th, etc., until some argument between the 17th and 33rd is found to exceed the test argument, or equal it. When this is found, the Table Search is discontinued and the following items are to be found:
If for any argument X_{test}, X_n is defined as the first argument greater than or equal to X_{test} and the previous argument X_{n-1} is less than X_{test}

Location		*Contents*
ARG1	(ARGument 1)	X_n
LAR1	(Location of ARgument 1)	Address of X_n argument
PAC1	(Pseudo-ACcumulator 1)	Corresponding function $f(X_n)$
ARG2	(ARGument 2)	X_{n-1}
LAR2	(Location of ARgument 2)	Address of X_{n-1} argument
PAC2	(Pseudo-ACcumulator 2)	Corresponding function $f(X_{n-1})$

All of these are useful as addresses, although LAR1 and LAR2 may be used only with the RPL operation, and none of the addresses is indexable. *Caution!* Arithmetic operations use PAC1 as a result address, so the contents of PAC1 after a TSC will have to be used before any arithmetic operation or else transmitted to a temporary location.

It should be standard practice to make the last argument in any table equal to the highest number in the PRINT system (+99+999999 . . . .). This dummy number ensures against overrunning the table with an unexpectedly high argument. In a table of 398 entries, for example, it would also be very practical to make the last 3 entries to be this dummy number, thus increasing

the number of arguments to 401 and allowing a gross search interval of 20, which is the most efficient.

Let $\triangle$ symbolize the number of words lengths between the table of arguments and the corresponding functions. If $\triangle$ were set to zero, PACi would not contain the functions of the argument; the contents would be identical to the contents of ARGi, which are the arguments again. Sliding sets of tables might be constructed with this feature, using a variable $\triangle$. Furthermore, using $+\triangle$ in one case and $-\triangle$ in another permits interchange of the dependent and independent variables.

A standard method of coding is shown in the example, illustrating Table Search and linear interpolation, according to the formula:

$$f(X_{test}) = f(X_{n-1}) + \frac{f(X_n) - f(X_{n-1})}{X_n - X_{n-1}} (X_{test} - X_{n-1})$$

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
	RPT	- 1, 0, (interval), 0	
	TSC	Δ, XSUB 1, XTEST	
	SUB	, PAC 2, TEMP 1	$f(X_n) - f(X_{n-1})$
	SUB	XTEST, ARG 2, TEMP 2	$X_{test} - X_{n-1}$
	SUB	ARG 1, ARG 2	$X_n - X_{n-1}$
	DIV	TEMP 2	(PAC 1 implied as divisor)
	PMA	PAC 2, TEMP 1, RSULT	$= F(X_{test})$

Non-indexable computing operations

Transfer operations

There are four conditional and one unconditional operations in this group. Conditional transfer commands are written with mnemonic symbol and two positions in the variable field. The address in the first position is always that of the instruction to which the transfer is to be made if the condition is met. The address in the second position is that of the number whose condition is to be tested. The mnemonic symbols TRZ, TNZ, TRP and TRM signify respectively that this condition is to be zero, non-zero, plus or minus. TRU signifies that transfer is to be made unconditionally to the instruction whose address is in the single position in the variable field. In the example shown, the program will operate in normal sequence if the contents of JONES is a positive non-zero number; otherwise control transfers to the instruction in B006 and proceeds sequentially from there.

SERIAL 1- -5	LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
03041		TRZ	B006, JØNES	
03042		TRM	B006, JØNES	
03043		SQR	JØNES, SMITH	

Replace operation

The instruction for this operation is written with the mnemonic RPL (RePLace) and two positions in the variable field. This operation causes the instruction in the address specified by the second position to have its first position address replaced by the first position address of the RPL instruction. If the first position address is written "LAR1" or "LAR2" the replacement address is not LARi but the address in LARi. This indirect address feature is used in conjunction with the TRC, TRE and TSC operations. This operation has three other usages. It may be used as a "flip-flop" or sequencing

switch, for direct exiting from sub-routines and for command modification by replacement rather than by indexing.

RPL will operate only on the arithmetic, mathematical function and data transmission operations, all transfer operations, ATR, FXP and FPR. In addition, it will operate on the transfer addresses of the WLi, WHi, RCD, WTi and RTi operations, although these addresses are not in the first position of the variable field. The example shown depicts the condition of the instruction in TEXAS before and after a RPL. Further examples of usage may be found in Appendix II.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
TEXAS	MAD	CØMIC, 1, SMITH, 2, RSLT3	
	RPL	CAPS, TEXAS	
TEXAS	MAD	CAPS, 1, SMITH, 2, RSLT3	(new form of TEXAS)

Extract operation

The instruction for this operation is written with the mnemonic XTP (eXTract Power) and two positions in the variable field. The first position contains the address of the floating point number whose power is to be extracted. The second position contains the address of another floating point number whose power is to be replaced by the power extracted from the first number. This operation is designed for convergence testing, since in floating point the size of a number during the course of calculation may not be predicted. The example shown illustrates the programming of a convergence test on (JANES), where it is desired that the valid value of (JANES) shall not differ from the previous value of (JANES) by more than 3 in the 7th digit of the mantissa. The power of (JANES) is assigned to mantissa of .3 in TEST and scaled by 10^{-6}. After step CONV2 the number in TEST is the proper value for testing for convergence. When the difference between the present and previous value becomes less than this increment, the iterative loop is abridged by the TRC command. Without such an instruction, an oscillatory condition in the last digit of an iterated number might make it impossible to exit from the loop. It also provides for a less exacting matching than all of the digits in the mantissa. An appreciable speed-up of computing time may

also be realized in slowly converging operations, if less stringent accuracies are made acceptable.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
TEST	FLC	-0+3	Power has no significance
SCAL E	FLC	-5+1	10^{-6}
	ENT		
RSTR T	TMT	JANES, SMITH	Send old JANES to SMITH
			Compute new value for JANES
			with proper expression
CØNV 1	XTP	JANES, TEST	
CØNV 2	MPY	TEST, SCALE, TEST	
CØNV 3	SUB	JANES, SMITH	Differential in PAC I
CØNV 4	TAB		Absolute value of differential
CØNV 5	TRC	RSTRT,, TEST	To RSTRT if not converged

Set index register operations

The instruction for this operation is written with the mnemonic SRi (Set Register) and two positions in the variable field. The third character in the mnemonic symbol is written 1, 2 or 3, thus specifying the number of the index register to be set. It is set to the number of plus or minus word lengths in the first position.

The limit tally of that register is set to the number of word lengths written in the second position. If the second position is blank, the limit tally will automatically be set to zero. The limit tally is always a positive quantity and when converted (by multiplying by data word length, which is 2 plus the mantissa length) must be less than or equal to the memory capacity of the 705 minus 10,000.

Non-test transfer operations

The instructions for these operations are written with the mnemonic TNi

(Transfer No test) and two positions in the variable field. The third character in the mnemonic symbol is written 1, 2 or 3, thus specifying the index register to be operated upon. The first position contains the address of the instruction to which unconditional transfer is made after augmenting the contents of the index register with the number of word lengths written in the second position. This transfer address will, in many cases, merely be the next instruction. The increment for the index register may be either plus or minus; the minus sign is written before the number and no sign is written for plus numbers.

Test transfer operations

The instructions for these operations are written with the mnemonic TXi (Transfer testing indeX) and two positions in the variable field. The third character in the mnemonic symbol is written 1, 2 or 3, thus specifying the index register to be operated upon. This operation functions in the same manner as TNi except that the transfer (to the instruction whose address is specified in the first position) is nullified if the contents of the index register, as now incremented, are equal to or greater than the limit tally previously specified. If this is so, the program does not transfer but rather proceeds to the next instruction in sequence.

Repeat operations

The instructions for these operations are written with the mnemonic RPT (RePeaT) or RWR (Repeat With Reset) and four positions in the variable field. RWR is equivalent to RPT except that PAC1 (the primary pseudo-accumulator) is reset to zero. A RPT signifies that the next instruction in sequence is to be repeated (i.e. performed) the number of times specified in the first position of the RPT. This instruction is to be performed as written the first time, but for each repetition the numbers or addresses in the first, second and third positions of that next instruction are to be additionally augmented by the numbers respectively in the second, third and fourth positions of the RPT. If the next instruction does not have a third address, it is not necessary to specify a fourth position for RPT.

RPT and RWR apply only to the next instruction, not to any sequence

of instructions. Their purpose is to both minimize the number of instructions written by the programmer and reduce operating time on repetitive instructions. This is accomplished by letting the executive routine know in advance that the next instruction is repetitive so that interpretation and command fabrication is performed only once.

Indexing by RPT is secondary and subordinate to indexing by the contents of index registers and the simultaneous use of both is possible. All four positions of the variable field may be written as 1 or 2 digit numbers and all may be signed both plus and minus. The first position, however, is normally plus, for when it is signed minus it indicates indefinite repeat and as such must be used with caution. Indefinite RPT is designed to be used with the TRC, TRE and TSC operations. When an exit is made for any of these, the repeat tally is automatically reset to zero so as not to influence the next instruction in sequence. A leading asterisk in any of the second through fourth positions indicates that incrementation will be by that integer number rather than by that number of word lengths. This is mainly used for indexing the card column on FLO and the decimal position in the type wheels for FXP.

All indexable instructions and only indexable instructions are repeatable. Each of these interrogates the number in the first position (serving as a count) before storing the result. If this is non-zero, the count is reduced by 1 and the operation is automatically repeated with further indexing by the RPT increments. If it is zero, it signifies either that the instruction was not intended to be repeated or that it has been performed for the last time. In either case, the program proceeds to the next instruction in sequence. In the case of arithmetic commands, if the address to which the result is sent is not indexed by the RPT, the result is stored intermediately in PAC1 only, and not sent to the result storage until the repeat tally is zero.

Advantage may be taken of the fact that RPT and RWR do not alter the contents of PAC1. The following example illustrates the calculation of $\left(\frac{A}{B}\right)^n$, n being an integer, which result is then available in PAC1.

LOCATION	OPERATION CODE	VARIABLE FIELD	COMMENTS
6- .10	11- .13	14-	.80
TEMP	REG		
	ENT		
	DIV	LØCA, LØCB, TEMP	
	RPT	(n-1)	
	MPY	TEMP	

Switching operation

The instruction for this operation is written with the mnemonic ATR (Alternating TRansfer) and two positions in the variable field, each of which is tagged. In operation, an unconditional transfer is made to the address in the first position each time the instruction is executed, up to the number of times designated by the tag for that position. After this limit is reached, succeeding executions of this instruction cause unconditional transfer to the address in the second position, up to the number of times designated by its tag. The instruction then reverts to the original condition for further alternation as required. Execution is dynamic and it is impossible to return to the initial condition without performing the entire cycle; thus a conditional transfer exit from the cycle destroys the utility of the ATR unless the program is read in again to restore the initial conditions.

Tags for both positions are unsigned positive numbers, from 1 to 400. A zero is an illegal tag for which pre-edit will substitute a 1. For purposes of counting only, this operation is generally more efficient than using index registers with their transfer instructions.

The first example shows the preferable, but not the only, method for executing n times the routine commencing with the operation in the address "START". The second example illustrates a method for simulating the general extended case when the desired tags for both addresses exceed the limit 400 and are not prime.

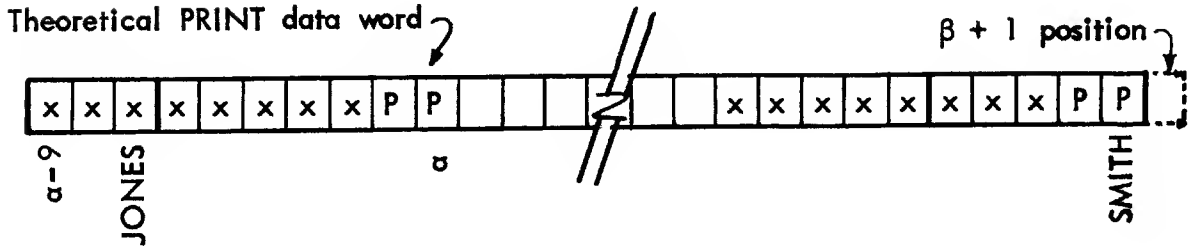
LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
STAR T			
	ATR	START, (n-1), NXTCM, 1	
NXTC M			
STAR T	ATR	FIRST, a, SECND, c	SIMULATES:
FIRS T			
			ATR FIRST, ab, SECND, cd
	ATR	FIRST, (b-1), START, 1	
SECN D			WHERE:
			ab > 400, NOT PRIME
	ATR	SECND, (d-1), START, 1	cd > 400, NOT PRIME

Generating print instructions

As programmers become more experienced in using the PRINT system, the translation charts on pages 51 and 52 will become increasingly useful. PRINT instructions, due to their variable length and specialized format for maximum operating speeds, may not be modified directly except by indexing. Very often specific coding for a problem will depend upon parameter values. An excellent example of this is the martix inversion kernel in Appendix II. Rather than recode the problem each time for a different order of matrix and a different number of column vectors, it would be advantageous to have 705 instructions preceding the general coding which would generate the necessary variable portions of the PRINT instructions, given only the values of n and b. To do this, the structure of the translated PRINT instructions must be known.

As an example, consider the tape instructions RTi and WTi, which are specifically designed to operate with PRINT data words in fixed lengths. The coding kernel below shows how to make use of these same instructions to

write and read irregular blocks of memory on tape, taking advantage of the error-correction routines contained in these PRINT instructions. The record as formed is illustrated below:



LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
JØNES	BLK	3	
	⋮		
SMITH	BLK	5	
BRØWN	ADC	JØNES - 2	α - 9 portion
BLACK	ADC	SMITH + 1,9	β + 1 portion
	LØD	BRØWN,4	
	UNL	PUTT + 12,4	
	UNL	TAKE + 12,4	
	LØD	BLACK,4	
	UNL	PUTT + 16,4	
	ENT		
PUTT	WT8	0000,0000	Zeros are dummy addresses, later replaced
	⋮		
TAKE	RT8	0000	Zeros are dummy addresses, later replaced

Fixed symbolic locations

Actual location for the fixed symbolic data words defined on page 4 are:

	Address of right-hand digit		
Mantissa:	*8-digit*	*10-digit*	*12-digit*
PAC1	0254	0256	0258
PAC2	0244	0244	0244
ARG1	2439	2441	2443
ARG2	2449	2453	2457
LAR1	2044	2059	2059
LAR2	2019	2019	2019

In addition, all line, heading and card image positions are considered fixed symbolic locations if addressed as:

TWxxx . . . . . TW stands for type wheel, xxx for 001 to 120
HDxxx . . . . . HD stands for heading, xxx for 001 to 120
CØLxx . . . . . CØL stands for column, xx for 01 to 80

Locations of these images are:

LINE	3265-3385	Address=TW+3265
≢	3386	
HDG	3387-3507	Address=HD+3387
≢	3508	
CARD	3509-3588	Address=CØL+3508
≢	3589	

The position referenced in the coding is considered as:

1. BADD for PRINT data words as called for in PRINT instructions.

   xxx. . . .$\overset{\pm}{x}\overset{\pm}{x}\overset{\pm}{x}$ $\overset{\pm}{P}\overset{\pm}{P}$ Pre-edit must make conversions of BADD as required for the several types of PRINT instructions.

2. The addressed character for 705 commands.

   Examples: UNL CØL06=UNL (3271)
   ST TW086+4, 15

Image addresses must contain all five characters. In the above examples, CØL6 or TW86 would not be allowable.

Input—output operations

Printing operations

Much emphasis has been placed, within the PRINT system, upon ease of entering data and recording results. Both on-line and off-line operations are equally feasible. There are several operations especially designed for simplified control of printed output.

FXP (FiX for Printing) and FPR (Fix for Printing Rounded) are indexable operations which will convert the floating point number in a specified address to a fixed decimal condition and store it in the specified position in the line image in memory just as it should be printed.

The variable field of these instructions consists of the address of the number to be converted, index tag (if any), a 1 to 3 digit type wheel position for the decimal point, a 1 or 2 digit maximum number of expected whole numbers and a W, the number of decimal positions and a D, and a 1 or 2 digit power of 10 by which the number is to be scaled. The last position should be left blank if there is no scaling. (The heading should note this scaling if it exists). Typical commands and results are:

FXP JONES, 28, 2W, 10D, —2 . . . . 123456787$\overset{+}{8}\overset{+}{0}$3	bl. 2345678780b
FPR G116, 13, 9, 4W, 2D . . . . . 43869261$\overset{-}{7}\overset{+}{8}$04	4386.93—
FPR G116, 3, 4W, 2D . . . . . 43869261$\overset{-}{7}\overset{+}{8}$04	86.93— (ERROR)

In the first example, 2, 20, and +20 would have been acceptable as scale factors. The error shown is due to calling for 4 whole numbers to the left of type wheel 3. Similar errors can occur by exceeding type wheel 120. Only the address in the first position is modifiable by index registers. Both the address and the type wheel position are modifiable by RPT or RWR. An example is shown where more than one number is converted to the same specifications; this will occur quite often in matrix output. In the example, (RØW01) are fixed with the decimal point in type wheel image 8, (RØW01+1) with the decimal point in type wheel image 19, etc., thus using only three instructions to convert the entire line and print it. 0W and 0D must be used to indicate absence of either whole numbers or decimals.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
	RPT	8, 1, *11	
	FXP	RØW01, 1, 8, 1W, 6D	
	WLS	PRINTER	

There are many combinatorial instructions for output control, as illustrated. All are non-indexable. Either the printer or a tape unit may be specified in the variable field. Either the full word description or the initials P or T may be used. An asterisk as the preceding character signifies a fast skip under carriage tape control.

Both on-line and off-line printers must be set to program control to use the PRINT 1 system. Except for special skip instructions, there should be a punch in only channel 1 of the carriage tape; a channel 12 punch is illegal and will cause an error message. All carriage control should be built into the program to eliminate most tape-changing.

Headings are put into the heading image with the HDG entries; for multiple usage of heading in a single run, reserve extra positions for heading components in memory with CON operations, and transmit these to the heading image as required. When a line is written successfully, without echo checks or other errors, the line image is erased to blanks. This allows a flexible line format, as the programmer makes provision for positioning only those numbers which he wishes to print, regardless of the make-up of the previous line. Card and heading images are NOT erased after writing.

WLN	PRINTER	Write a Line - No spacing
WLS	PRINTER	Write a Line - Single spacing
WLD	TAPE 4	Write a Line - Double spacing
WHD	T 7	Write Heading - Double spacing
WHT	P	Write Heading - Triple spacing
WL 2	TAPE 7	Write a Line - Skip to channel 2 punch
WH 4	* TAPE 8	Write Heading - Fast skip to channel 4

These operations may have counting transfers added by using two more positions in the variable field. The second position is a 1 or 2 digit number specifying the number of times the write line or heading operation is to be

executed, up to a limit of 98. The third position contains the address of the instruction to which control is to be transferred when writing is attempted after the limit is reached. This transfer restores the initial condition of the instruction so that the same process may be repeated. The following example shows the control operations for writing 20 pages, each with a heading and 50 lines of answers, grouped in 5 groups of 10.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
HEAD	WHT	T6, 20, PAGES	PAGES, LINES and LASTL are used
CØMP U	— — —	- - - -	as convenient mnemonic names for
	WLS	T6, 9, LINES	the associated instructions. The first
	TRU	CØMPU	line therefore reads:
LINE S	WLD	T6, 4, LASTL	
	TRU	CØMPU	"Write a Heading, Triple space,
LAST L	WLI	T6	on Tape 6 - write 20 PAGES."
	TRU	HEAD	
PAGE S			(continues computation after 20 pages are written)

Tape data storage operations

WTi (Write Tape) and RTi (Read Tape) are indexable operations provided for storage and retrieval of data words on tape, which is essentially a function of increasing memory capacity. The third character of the mnemonic symbol is the dial setting of the tape unit addressed. WTi is written with two positions in the variable field, each of which may have an index register tag. These positions contain the first and last addresses of a consecutive series of words in memory which are to now constitute a record on tape. RTi is written with one position in the variable field, which is the starting address in memory for reading one record from tape. As many words will be replaced in memory as the record itself contains, so the programmer is cautioned to know the pattern of his tape operations very thoroughly, to avoid destruction of wanted data. Discretion should also be maintained in using these instructions with the 4 tape units normally associated with pre-edit and library.

It is possible to add other positions in the variable field of WTi and RTi.

The third position may be a transfer address or the two letters TM. The fourth position must be TM, and exist only if there is an address in the third position. The transfer address of WTi is that of a sequence of instructions defining procedure in case the physical end of tape is reached before completing the write instruction. TM puts a tape mark as the next record after completion of writing. The transfer address of RTi is that of a sequence of instructions defining procedure in case where the record consists of a tape mark written by a WTi.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
	WT6	B001, B020	
	WT6	B001, 1, B020, 12, PATCH	
	WT5	G136, 1, G136, 12, TM	
	RT5	RAND	
	RT9	FIRST, TRØUT	
	BS8	20	(backspace tape 8 by 20 records)
	RW8		(rewind tape 8)
	TM8		(write a tape mark on tape 8)

BSi (BackSpace tape) and RWi (ReWind tape) are non-indexable operations for positioning records to be read or written. In the variable field of BSi is written the number of records to be backspaced. This is a 1 to 3 digit number; the programmer may not specify more records than exist on the tape from that point back. RWi has no information in the variable field, nor does TMi, which is a separate instruction for writing a tape mark unconnected with other operations.

Card operations

The card image in memory is used for both reading and punching. Special facilities are provided for reading both floating point and fixed point data, but punching is restricted to floating point form unless special handling is made in 705 language. This is based on the assumption that actual punched cards will be produced for local re-loading only, in which case there is no

purpose in refloating data which may be had already in floating form. Suggested floating point loader cards are as follow:

8 digit mantissa	8 words per card	addressed at 10(10)80
10 digit mantissa	6 words per card	addressed at 20(12)80
12 digit mantissa	5 words per card	addressed at 24(14)80

The card for the 8 digit mantissa may be reduced at option to 7 words, thus allowing the first 8 columns in any system to be indicative information. For fixed data, there is no specified format and commands are so designed that it is not necessary. Typical movement of data and production of a punched card might be:

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
	RPT	7, 1, *10	
	TMT	A001, CØL20	
	WCD	T4	

WCD (Write CarD) and RCD (Read CarD) are operations for reading and writing 80 character records. Reading may be from either the card reader or a numbered tape unit, and this is written in the variable field. Writing is onto either the punch or a numbered tape unit, and this is specified in the variable field. The use of tape for these operations is designed for peripheral equipment, although it is another method of temporary data storage in fixed decimal format. The unit in the variable field may be written with the full name or the initials T, P or R as required. The transmission to or from the card image in memory is implicit in all of these instructions. The second position of RCD is a transfer address for an end-of-file condition.

LOCATION	OPERATION CODE	VARIABLE FIELD	COMMENTS
5- -10	11- -13	14-	-80
	WCD	TAPE 8	
	WCD	PUNCH	
	WCD	P	
	RCD	T6	
	RCD	READER, TRADD	

FLO (FLOat) is an operation for converting a fixed point number of any specified length to floating point form, thus making it suitable as an operand in the PRINT system. It is written with these four positions in the variable field:

1. The symbolic address of the units position of the number to be converted. This will most often be CØLxx. A sign for this number must exist over the units position for negative numbers only.

2. The number of digits comprising the number. Must be <2 mantissa lengths.

3. The direction (L or R) for shifting the decimal point to put it in the true position and the number of places to shift, considering the number to be originally comprised of whole numbers. (See examples). For no shift, either L0 or R0 must be coded.

4. The symbolic location where the number is to be stored after conversion, with an index register tag if required.

Only the address in the fourth position is indexable by the contents of index registers, but both it and the address in the first position may be indexed by a RPT instruction. The first position address will most commonly be CØLxx, and the RPT increment will most commonly be asterisked to indicate number of character positions rather than word lengths. The indications in the comments field of the examples show the true decimalization of the numbers. The third example is a program for reading in 200 pieces of 4 digit data for processing, condensed for loading purposes on 10 cards. When converted to floating point form, the data words occupy PRINT memory addresses A001 to A200 inclusive.

SERIAL 1- -5	LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
		RPT	4, 1, 1	
		FLØ	FXWRD, 6, R2, INPUT, 3	XXXXXX00.
		RPT	6, *5, 1	
		FLØ	CØL48, 5, L2, TAX4	XXX.XX
01010	A200	REG	A001	
01020		SR 1	0, 200	
01030	RDATA	RCD	READER	
01040		RPT	20, *4, 1	
01050		FLØ	CØL04, 4, L3, A001, 1	X.XXX
01060		TX 1	RDATA, 20	

Pre-edit and system entry

If a system tape does not exist by virtue of previous usage of the PRINT 1 system, operations are commenced by placing the PRINT 1 program deck in the card reader, followed by symbolic cards for the program to be pre-edited. The system is then initiated from the console by:

	Address selector	*Typewriter key*
1. Clear memory		
2. Place in manual instruct status		
3. Select the card reader	0100	2
4. Read into lowest memory position	0000	Y
5. Depress the start key		

After loading these cards, the PRINT 1 system will be on tape 0200, in 9 sections:

1. System control 1
2. Memory print (13 records)
3. Tape print
4. First pass of pre-edit
5. Intermediate pass of pre-edit
6. Last pass of pre-edit
7. System control 2
8. PRINT 1 executive routine
9. Non-standard library

If a system tape does exist, it is loaded on tape 0200, and the system initiated from the console by:

	Address selector	*Typewriter key*
1. Clear memory		
2. Place in manual instruct status		
3. Select the system tape unit	0200	2
4. Rewind the system tape	0002	3
5. Turn off IOF	0000	3
6. Read into lowest memory position	"	Y
7. Depress the start key		

Programs may be pre-edited from card, tape or combined card and tape input. Tape input will be on 0202 if Alteration Switch 0915 is OFF, on 0203 if it is ON. The combined card and tape input feature exists for purposes of repairing programs. Steps may be inserted or deleted by change cards. A complete reorganization of the program within memory takes place every time this is done.

Pre-editing starts with system control 1 and proceeds to the first pass. Card and tape input are checked for sequence and merged on serial number (col. 1-5 of the card). When the serial of a card matches that of a card image on tape, the card record replaces that tape record unless the card carries the mnemonic operation code DEL (DELete). In this case, that record is deleted and so are all subsequent records up to and including the record whose serial is punched in the variable field of the DEL card. Tape records having DEL for operation and DEL cards without matching tape records are both deleted. Non-DEL cards whose serials do not match with any card images on tape are collated with them.

The output of the first pass is on 0202 or 0203. Records contain the actual locations of the instructions. Other conversion is deferred until the last pass. In the event that the assignment table overflows available memory, the overflow blocks will be on 0201. The intermediate pass is executed only when 3 or more overflow blocks occur; this pass finds actual addresses for the symbolic addresses referring to the overflow blocks. When there are 3 or less blocks of the assignment table yet to be searched, the last pass is called into memory for operation. This last pass completes conversion of the mnemonic intructions, writing:

1. A program tape on 0201, consisting of actual 705 instructions and converted pseudo-instructions. If Alteration Switch 0913 is ON, standard 705 load cards will be punched for reloading by card rather than the 0201 tape. This is

suitable for short programs, or where a permanent record of the program is desired for storage in a more flexible medium. This should be done only when the program is known to be correct and working.

2. A master symbolic tape on 0203 (or 0202 for referrals > 1000) which contains the corrected and updated program in symbolic form, just as the original punched cards were. This is suitable as input for re-editing and further correction. The selection of either 0203 or 0202 as the proper input tape for re-editing is automatic. A concluding typewriter message will indicate the correct location of tapes.

3. A listing tape on 0202 (or 0203 for referrals > 1000) which is the permanent record of coding, pre-editing and assembly of the program. If Alteration Switch 0914 is ON, the listing will be written on the line printer during the pre-edit process, in which case this tape need not be saved unless more copies are desired. If the switch is OFF, the availability of an auxiliary printer is normally assumed. For both printing methods, time will be lost if the comments exceed 25 characters, since an extra line will be printed just to accommodate the overflow. Comment characters above 50 will not be printed on this listing.

If the program is to be executed without pre-editing, Alteration Switches 0911 and 0912 are both OFF, thus diverting to system control 2. Alteration Switch 0913 is then interrogated. If it is ON, the edited program will be read from punched cards; if it is OFF, the program will be read from tape 0201. The combination of both card and tape input is not possible here. System control 2 reads the edited program and the executive routine into memory, activating a typewriter message calling for setting Alteration Switches for the program to be executed and stopping on HLT 1111. Depressing the start key will cause execution of the program starting at the first instruction, which is either a 705 instruction or the compiled ENTer instruction in PRINT. If the 0902 indicator is turned on, loading is in error. Press the start key to reload from tape. Cards must be reloaded; reset, start and read again. Pre-edit will blank unused memory before reading in the program.

Memory print and tape print routines are incorporated in the system tape. They are called into use by setting Alteration Switch 0916 ON and depressing the reset and start keys. A typewriter message will give instructions for next setting of 0916 (OFF to bypass memory print). The tape to be printed is selected by setting Alteration Switches 0911 thru 0914 in a binary representation of the units position of the tape unit desired, as:

Alteration switch	*Value if ON*	*Value if OFF*
0911	8	0
0912	4	0
0913	2	0
0914	1	0

For example, if 0912 and 0914 were the only switches ON, it would signify that tape 0205 would be printed, as 4 + 1 = 5. Configurations which sum more than 9 are in error. Printing of the tape selected continues until a tape mark is sensed or operation is changed to manual. Tapes may be selected and printed successively but in any order, by changing the Alteration Switch configuration and depressing the start key each time. The return to system control 1 is effected by turning 0916 OFF, reset and start.

Pre-edit conversion

Two types of addresses are recognized by pre-edit. The basic address of a floating point data word is that of its highest (or right-hand) memory position. Pre-edit allocates memory in $(m + 2)$ modules, where m is the mantissa length. FLC and REG are the two operations which cause memory to be reserved this way.

The basic address of a PRINT pseudo-instruction, which is of variable length, is that of its lowest (or left-hand) memory position. BADD = (the basic address of the previous instruction) + (the length of the previous instruction), since they are normally obeyed in order of ascending memory position.

When either REG or FLC is encountered by pre-edit, a test is made to see if the previous instruction was either REG or FLC. If not, (and a previous ORG falls in this category), the location counter leaves a blank preceding the entry to insure definition of a numeric field. If an initial origin is supplied in the program it will take precedence over the standard origin supplied by pre-edit, which follows immediately after the executive and loading routines.

The typewriter may operate during pre-edit to send error messages about system restrictions which have been ignored in coding. Each message is identified with the serial and symbolic location of the erroneous instruction. Some of these are for:

1. RPT or RWR tally > 99.

2. Non-repeatable instruction following a RPT or RWR.
3. Actual address for 705 instructions TR and 00 TMT not ending in 4 or 9.
4. Infraction of rules for symbolic or actual location addresses.
5. Minus index limit for any register, or a converted limit > (memory—10,000).
6. More than 2 HDG cards.
7. Problem overflows memory.
8. Non-PRINT or non-705 operation codes.
9. Attempting to increment non-indexable address (i.e. PAC1, PAC2, etc.)
10. ATR tally greater than 400.
11. Non-indexable entry tagged (i.e. PAC1, decimal location in FLO, etc.)

There may be instances when the programmer has a definite and legitimate purpose in ignoring these restrictions. Error messages do not necessarily indicate that revision must be made; they exist to warn the programmer to be certain that this was his true intent.

When a floating sub-routine symbol is coded, the pre-edit knows that the symbol has no assigned operation code number in the table of correspondence. The operation code for all floating sub-routines is assigned to it (this code comes from the last two digits of the address of the first instruction in the FSR area). Pre-edit automatically compiles the 705 instructions necessary to bring the proper sub-routine (if it exists on the library tape 0200) into the floating position in PRINT during the course of computation. Such a linkage is compiled only the first time that function is needed, or if another function has superseded it before it was to be used again. The criterion for compiling the linkage is thus change of requirement only. If only one non-standard sub-routine is used for a particular problem, the net effect is as though it were a permanent component of the executive routine in memory.

No floating sub-routines will be furnished with this manual. They are primarily the responsibility of the user, although IBM will distribute any routine contributed. The "tinkertoy" appendix in the supplement will show various means of extending this feature so that the programmer may specify the amount of memory he is willing to expend for floating sub-routines. Replacement would then be set up only if the desired sub-routine exceeded in size the amount of available specified memory left.

Summary—System operation

Tape Assignments 0200—System Tape 0202—Listing
0201—Actual Program 0203—Symbolic (updated)

Function	*Operation*	*Alteration Switches* 0911	0912	0913	0914	0915	0916[1]
INPUT	Card and tape 0202	ON	ON			OFF	OFF
	Card and tape 0203	ON	ON			ON	OFF
	Card	ON	OFF				OFF
	Tape 0202	OFF	ON			OFF	OFF
	Tape 0203	OFF	ON			ON	OFF
OUTPUT . . .	On-line printer listing[2]	—	—		ON		
	On-line punched program[2]	—	—	ON			
	Memory and tape print[3]	8	4	2	1		ON-(ON)
	Memory print only[4]						ON-ON
	Tape print only[5]	8	4	2	1		ON-OFF
PROGRAM . .	Start key[6]						
	Start key	(As required for subject program)					
PROGRAM . . (Without pre-edit)	Card-loaded program	OFF	OFF	ON			OFF
	Tape-loaded program	OFF	OFF	OFF			OFF

[1] Alteration Switch 0916 must be OFF if re-entry to system is by reset and start, except as noted under memory and tape print instructions.

[2] These will be on tapes 0202 and 0201 respectively, regardless of settings.

[3] Reset-start. Start again after typed message. Memory print will occur first, then a tape print for each start until 0916 is turned OFF. Select tape units by binary representation, in 0911 thru 0914, of units digit of desired unit.

[4] Reset-start. Start again after typed message. Turn 0916 OFF, reset and start to return to system control 1.

[5] Reset-start. Turn 0916 OFF after typed message. Select tape unit by Alteration Switch combination. Selected tapes will print for each start until reset.

[6] Brings executive routine and subject program into memory to operate. After typed message and HLT 1111, set switches as required and depress start key.

Appendix I—Operation execution times

The execution times for certain operations are given here to indicate the general speed range for the PRINT 1 system, in running time. It should be noted that these times cannot reflect elapsed problem time, and that in general they bear the burden of flexibility and convenience. As the system is still in process, final times for other mantissa lengths than those shown here may be expected to vary. The final manual will contain a complete list of guaranteed execution times for all operations not associated with input-output equipment.

The times given here are for complete multi-address operation and include all interpretation and miscellaneous times. All times are given, in milliseconds, for a 10 digit mantissa system, except as noted for 8 digit.

	Non-zero operands		*Zero operands*	
	Single execution	*10 time average*	*Single execution*	*10 time average*
ADD, SUB	4.9	4.2	3.1	2.4
MPY, MMY	7.1	6.3	4.0	3.2
DIV, MDV	18.6	17.5	3.1	2.0
PMA, MPM	8.6	7.4		
MAD, MMA	8.8	7.6		

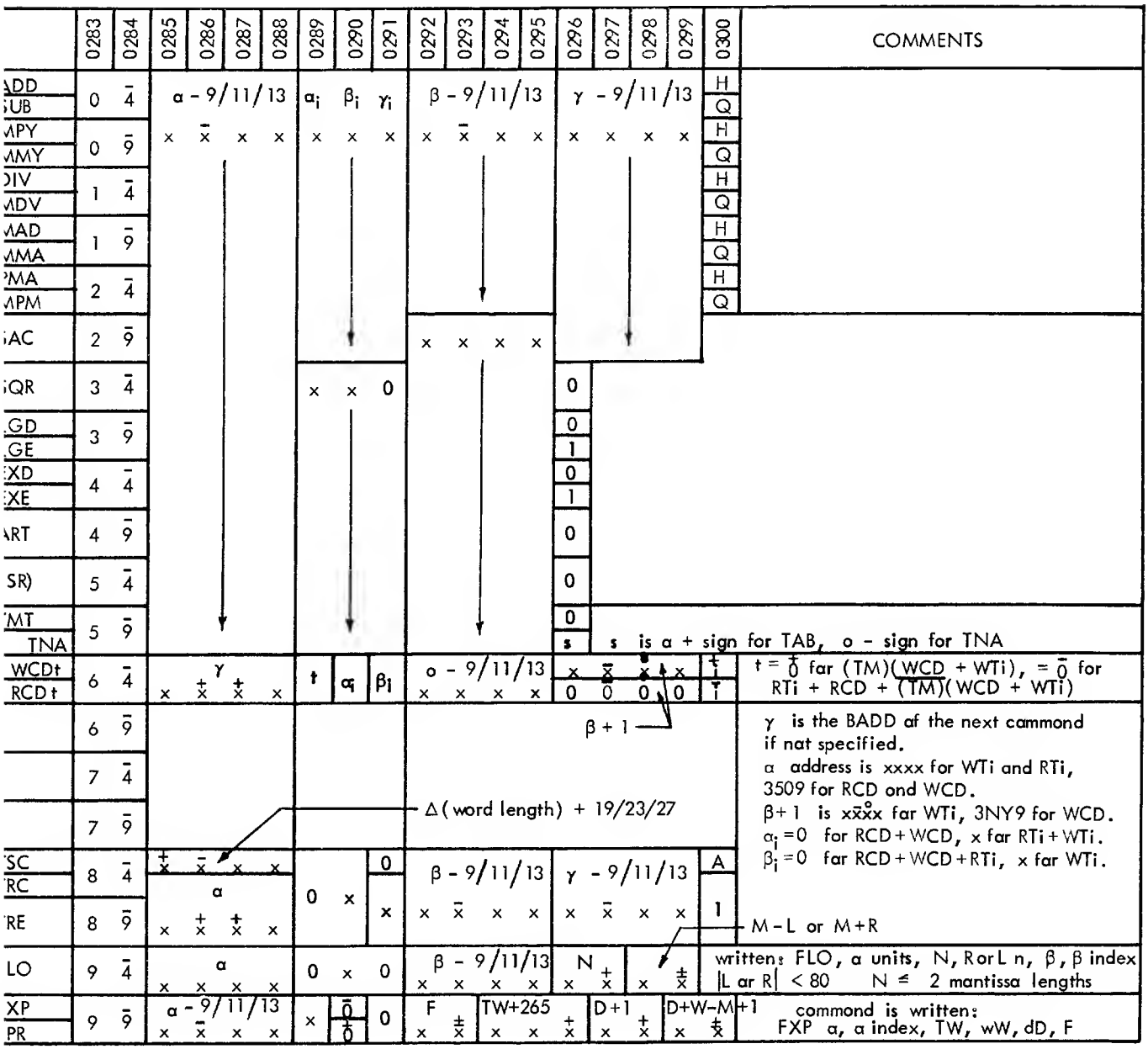
SQR	16.4 (8 digit)
ART	26.9 (8 digit)
EXE	20.1 (8 digit)
EXD	18.0 (8 digit)
SAC	25.7 (8 digit)
LGD	13.9
LGE	16.8

ENT	2.2	TMT	1.9
LVE	.8	TAB, TNA	2.1
TRU	.8	XTP	1.1
TRZ, TNZ, TRP, TRM	1.4	RPL	1.7
TRC	2.3	SRi	1.3
ATR	1.5	TXi, TNi	1.0
RPT, RWR	1.3		

Translation chart for non-indexable (non-repeatable) operations

	0283	0284	0285	0286	0287	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	COMMENTS
	0	+ 4	This operation sacrificed for system control.																
	0	+ 9																	
	1	+ 4																	
TRP TRM TRZ TNZ	1	+ 9	α x ± x ± x x				β − 2 x x ± x x				M N								Tens position of β − 2 is signed + for TRP and TRZ, signed − for TRM and TNZ
TRU	2	+ 4																	
RPT	2	+ 9	n − 1 ± x x		i x x x x			j x x x x				k x x x x				‡			n − 1 is either $\pm_{xx}$ or $0_{\bar{1}}$ for indefinite RPT. i, j and k follow the rules for SRi except = actual number only when asterisked.
RWR	3	+ 4																	
TN1 TX1	3	+ 9	o x x x x				1 2	Increment x x x x											o = basic address (BADD) of the command to which transfer is made. Increment follows the rules for SRi.
TN2 TX2	4	+ 4					1 2												
TN3 TX3	4	+ 9					1 2												
RPL	5	+ 4	α − 9/11/13 x x x x				β + 2 x x x x												β = command basic address. If RPL is indirect for LAR 1 or LAR 2, the α position is replaced by $20_{\bar{4}}$ or $20_{\bar{9}}$, respectively.
XTP	5	+ 9	α − 1 x x x x				β − 1 x x x x												α and β are basic addresses of data words.
ATR	6	+ 4	α x x x x		i x x		β x x x x				j x x		tally 0 0						i and j are zoned in the 10s position for count up to 399, in ADM collating sequence.
WLi WHi	6	+ 9	ar 0 0 0 0 (if not specified)		LC+1 x x		tally 0 0		unit x 0	±	2 6 N 3 8 G		u	s					s is the spacing control character. u is 4 for tape, 5 for printer. UNIT is 20i or 400, with units pos. zoned − for triple space, + for none, single or double. Line count (LC+1) is $00_{\bar{}}$ if not specified.
BSI RWI TMI	7	+ 4	i 3 0 0 0				D B A	n n h 0 0 A											
WCD punch RCD reader	7	+ 9	0 0 0 0 x x x x				3 1	0 0		R 9 Y 4									
SR1 SR2 SR3	8	+ 4 + 9 + 4	− limit x x x x				‡	set to x x x x											set = f(first position), − limit = f(second position). Both are unsigned true (or 40,000 complements) products of (number) times the (word length).
LVE	9	+ 9	1	α x x x x															α is first following 705 command location if not specified.

Translation chart for indexable (repeatable) operations



Appendix II—Example I

Generalized matrix multiplication

The coding symbolized here is a basic method to effect the multiplication of a (k by m) matrix and an (m by n) matrix to produce the product matrix (k by n). It is valid when sufficient memory locations can be reserved for the elements of all three matrices. Many of the features of PRINT 1 as applied to repetitive operations are illustrated in this general method.

The common practice in assigning memory to the elements of a matrix is to store them row-by-row in sequential addresses. In this case these sequential addresses are regional for convenience. The general elements of these matrices are to be in locations defined as:

Matrix	*Location*
(k by m)	A001 + (row−1) (m) + (col−1)
(m by n)	B001 + (row−1) (n) + (col−1)
(k by n)	C001 + (row−1) (n) + (col−1)

The program is generally written as below, with the proper numbers replacing k, m and n. All operations are shown with referrals in the location column, but steps 3, 4 and 8 are the only ones which need it.

LOCATION 5-10	OPERATION CODE 11-13	VARIABLE FIELD 14-	COMMENTS -80
STEP 1	SR 1	0	
STEP 2	SR 2	0, km	
STEP 3	SR 3	0, n	
STEP 4	RWR	m, 1, n	
STEP 5	MAD	A001, 2, B001, 3, C001, 123	
STEP 6	TX 3	STEP 4, 1	
STEP 7	TN 1	STEP 8, (n - m)	
STEP 8	TX 2	STEP 3, m	

If either of the two matrices to be multiplied are square, it should be used as the multiplicand. Under these conditions the 1st and 7th steps may be eliminated because m = n. Elements are computed row-wise in the example, for efficient storage, but the problem could be re-coded to develop them

column-wise. Printing of a row at a time during calculation is possible by inserting the necessary operations between steps 6 and 7, or 7 and 8.

A coding kernel is given here for matrix multiplication using tapes for input of elements. This is again for the multiplication of a (k by m) matrix by a (m by n) matrix. The (k by m) matrix is assumed to be stored on tape 0207 in k records, each record consisting of the m elements of a row. The (m by n) matrix is assumed to be stored on tape 0208 in n records, each record consisting of the m elements of a column. The first row and column are the first records of the respective tapes, etc. The product matrix is to be stored in row records on tape 0209.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
RØWS	RT7	A001	m elements in each of k rows
	SR2	0,n	
CØLS	RT8	B001	m elements in each of n columns
	RWR	m,1,1	
	MAD	A001,B001,C001,2	
	TX2	CØLS,1	
	WT9	C001,C00n	
	RW8		
	ATR	RØWS,(k-1),NXTCM,1	
NXTC M			

Appendix II—Example II

Three examples of efficient coding for polynomial evaluation are illustrated below. As a general rule, requiring more instructions than are shown here will indicate inefficient assignment of memory for arguments and coefficients. The programmer who has occasion to use this type of coding may profit by extension of these examples.

1. *Evaluating a Single Polynomial at N Arguments (N = 14)*

$$P(X) = \sum_{0}^{6} a_i X^i$$

Arguments X located in K203 (2) K229
P (X)s to be sent to K204 (2) K230
Coefficients a_i in (D006 + i)

2. *Evaluating N Polynomials at a Single Argument (N = 5)*

$$P_j(X) = \sum_{0}^{6} (a_j)_i X^i$$

Argument X located in K203
P_j (X)s to be sent to (D013 + 8j)
Coefficients $(a_j)_i$ in (D006 + i + 8j)

3. *Evaluating the Bi-variate Surface*

$$Z = \sum_{i=0}^{i=4} a_i Y^i \quad \text{where } a_i = \sum_{j=0}^{j=3} b_{ij} X^j$$

a_i are in (C005 + 5i), b_{ij} are in (C001 + 5i + j), X is in C026, Y is in C027 and the answer Z is to be placed in C028.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
	SR 1	0, 28	
NXTA R	RWR	7, -1	
	PMA	D012, K203, 1, K204, 1	
	TX 1	NXTAR, 2	
	SR 1	0, 40	
NXTP N	RWR	7, -1	
	PMA	D012, 1, K203, D013, 1	
	TX 1	NXTPN, 8	
	SR 2	0, 25	
NXTC Ø	RWR	4, -1	
	PMA	C004, 2, C026, C005, 2	
	TX 2	NXTCØ, 5	
	RWR	5, -5	
	PMA	C025, C027, C028	

Appendix II—Example III

Matrix inversion

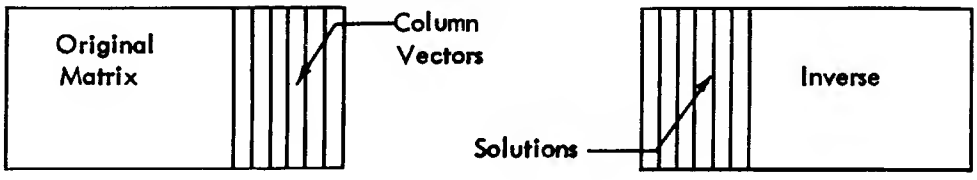
The coding kernel below gives a simple method for the inversion of a matrix and solution of simultaneous equations. It is adopted from R. DeSio's 650 program, using Jordan's method. The nth order matrix with b column vectors furnishes the array:

$$\begin{array}{cccccccccc}
a_{11} & a_{12} & a_{13} & a_{14} \ldots\ldots\ldots\ldots a_{1n} & V_{11} & V_{12} & V_{13} \ldots\ldots\ldots\ldots V_{1b} \\
a_{21} & a_{22} & a_{23} & a_{24} \ldots\ldots\ldots\ldots a_{2n} & V_{21} & V_{22} & V_{23} \ldots\ldots\ldots\ldots V_{2b} \\
\vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
a_{n1} & a_{n2} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots a_{nn} & V_{n1} & V_{n2} \ldots\ldots\ldots\ldots\ldots\ldots V_{nb}
\end{array}$$

This array is stored row-wise in memory from A001 to A000 + n(n + b), each row starting at A001 + (row − 1) (n + b). A001 + n (n + b) thru A000 + (n + 1) (n − b) are reserved as working storage, for a total of (n + 1) (n + b) words.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
REDUC	DIV	LØC 1, A001, A000 + (n+1)(n+b)	(Reciprocal of first element)
	RPT	(n+b-1), 1, 0, 1	
	MPY	A002, A000+(n+1)(n+b), A001 + n(n+b)	
	SR 1	0, (n-1)(n+b)	
UPRØW	RPT	(n+b), 0, 1, 1	
	MMY	A001+(n+b), 1, A001 + n(n+b), A001, 1	
	RPT	(n+b-1), 1, 1, 1	
	ADD	A002 + (n+b), 1, A001, 1, A001, 1	
	TX 1	UPRØW, (n+b)	
	RPT	(n+b), 1, 1	
	TMT	A001 + n(n+b), A001 + (n-1)(n+b)	
	ATR	REDUC, (n-1), NXTCM, 1	(Counting control)
NXTCM			

n reductions are required, giving a new array in the identical block of memory positions, according to the following schematic:



Orders of matrices which may conveniently be inverted in memory are:

Memory size	*8-digit mantissa*	*10-digit*	*12-digit*
20,000	38	35	33
40,000	58	53	49

A 10th order matrix may be inverted in 10 seconds, 20th in 1 min., 20 sec., and 25th in 2 min., 36 sec. The 50th order in memory takes 20 min., 40 sec., and approximately 25 min. on tape. The program kernel for inversion of a matrix stored on tape is shown below. Each record on tape 0208 is a row of the combined array. As shown, the coding is limited to 99th order because of the RPTs; insertion of multiple RPTs will increase the order capability.

LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
RTAP 8	RT8	A001	
	ATR	RØW1, 1, RØWN, (n-1)	
RØW1	DIV	LØC1, A001, A000 + 3(n+b)	
	RPT	(n+b-1), 1, 0, 1	
	MPY	A002, A000 + 3(n+b), A001+2(n+b)	
	ATR	RTAP8, 1, RTAP9, 1	
RØWN	RPT	(n+b), 0, 1, 1	
	MMY	A001, A001 + 2(n + b), A001 + (n + b)	
	RPT	(n + b - 1), 1, 1, 1	
	ADD	A002, A001 + (n + b), A001 + (n+b)	
	ATR	WTAP9, (n - 1), WTAP8, (n - 1)	
WTAP 9	WT9	A001 + (n + b), A000 + 2(n + b)	
	ATR	RTAP8, (n - 2), NRØW1, 1	
NRØW1	WT9	A001 + 2(n + b), A000 + 3(n + b)	
RWTP S	RW8		
	RW9		
	ATR	RTAP9, 1, RTAP8, 1	
RTAP 9	RT9	A001	
	ATR	RØW1, 1, RØWN, (n - 1)	
WTAP 8	WT8	A001 + (n + b), A000 + 2(n + b)	
	ATR	RTAP9, (n - 2), NRØW2, 1	
NRØW2	WT8	A001 + 2(n + b), A000 + 3(n + b)	
	ATR	RWTPS, n, NXTCM, 1	

Appendix II—Example IV

This program is to prepare a table of (X + cos Y)(X — sin Z), Y and Z being radian arguments. Each line is for 1 value of X, each page for 1 value of Z. Columns are for Y.

SERIAL 1- -5	LOCATION 6- -10	OPERATION CODE 11- -13	VARIABLE FIELD 14-	COMMENTS -80
01010		HDG	Housekeeping. The line descriptions and head-	
01020		HDG	ings are not coded for purposes of this example.	
01030	R008	REG	R001	Reserve
01040	SINE Z	REG		working
01050	TEMP	REG		storage
01060	X020	REG	X001	Reserve storage
01070	Y008	REG	Y001	for loading
01080	Z010	REG	Z001	input data
01090		ENT		
01100		RCD	READER	Read 2 data cards. The first con-
01110		RPT	20, *4, 1	tains 20 values of X as (xx.xx),
01120		FLØ	CØL04, 4, L2, X001	stored in X001 to X020. The sec-
01130		RCD	READER	ond contains 8 values of Y as
01140		RPT	8, *5, 1	(xx.xxx), stored in Y001 to Y008,
01150		FLØ	CØL05, 5, L3, Y001	and 10 values of Z as (x.xxx),
01160		RPT	10, *4, 1	stored in Z001 to Z010.
01170		FLØ	CØL44, 4, L3, Z001	
01180		SR3	0, 10	Control for number of pages (Z)
01190	PAGE	WHT	TAPE4	Heading, 2 spaces before lines
01200		SR1	0, 20	Control for number of lines (X)
01210		SAC	Z001, 3, SINEZ	Compute sinZ for each page
01220		TRU	RSETY	
01230	LINE	WLS	TAPE4	Write single-spaced result line
01240	RSET Y	SR2	0, 8	Control for number of columns (Y)
01250		SUB	X001, 1, SINEZ, TEMP	X - sinZ in TEMP (for each line)
01260	CØLMN	SAC	Y001, 2	cosY in PAC2
01270		ADD	X001, 1, PAC2	X + cosY in PAC1
01280		MPY	,TEMP, R001, 2	R001 through R008
01290		TX2	CØLMN, 1	Re-cycle inner loop
01300		RPT	8, 1, *11	Fix for print rounded the line of 8
01310		FPR	R001, 8, 4W, 2D	values, R001 to R008 (xxxx.xx±)
01320		TX1	LINE, 1	To write present line, ≠ 20th
01330		WL1	TAPE4	Write 20th line, skip over fold
01340		TX3	PAGE, 1	to new page on channel 1 control
01350		LVE	To 705 HLT on completion of 10th page.	

INTERNATIONAL BUSINESS MACHINES CORPORATION, 590 MADISON AVE., NEW YORK 22, NEW YORK

32-7334-1

Litho in U.S.A.